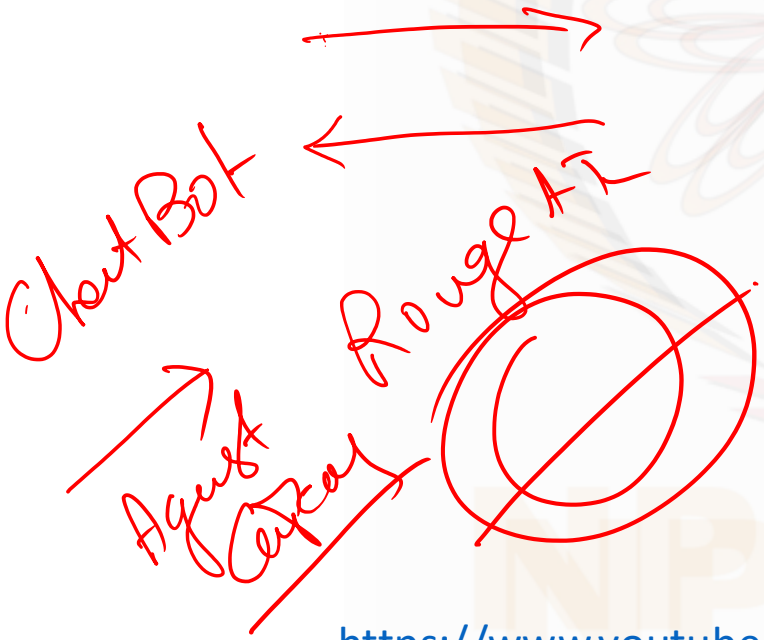
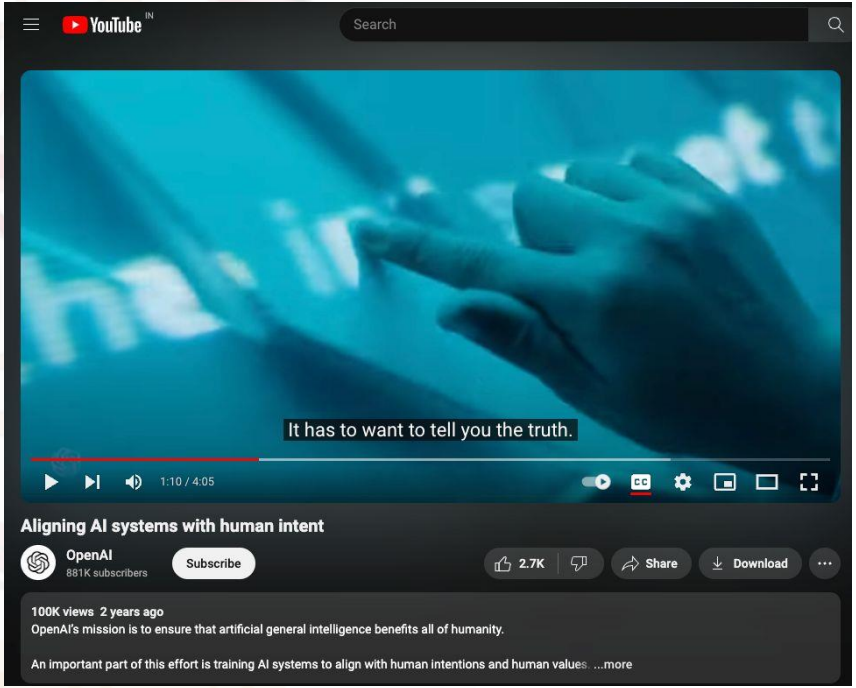
Responsible & Safe AI

Prof. Ponnurangam Kumaraguru (PK), IIITH
Prof. Balaraman Ravindran, IIT Madras
Prof. Arun Rajkumar, IIT Madras

Interpretability / Transparency / Probing

What is an alignment problem?





https://www.youtube.com/watch?v=yWDUzNiWPJA

Motivation

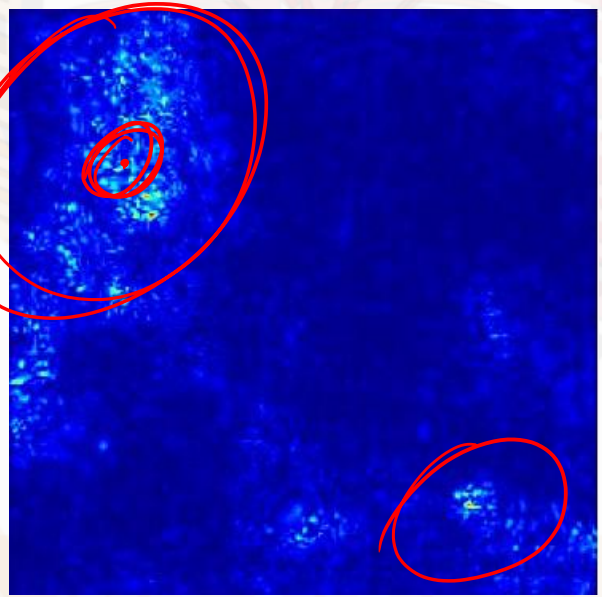
Transparency tools try to provide clarity about a model's inner workings

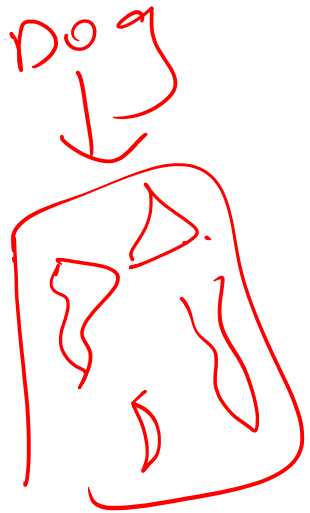
Model changes can sometimes cause the internal representations to substantially change, so we would like to understand when models process data differently

Transparency could make it easier for monitors to detect deception and other hazards

Pixel attribution methods

Highlight the pixels that were relevant for a certain image classification by a neural network





pixels are colored by their contribution to the classification

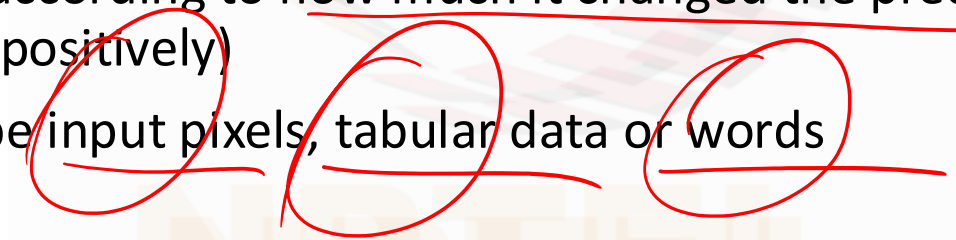
Pixel attribution methods

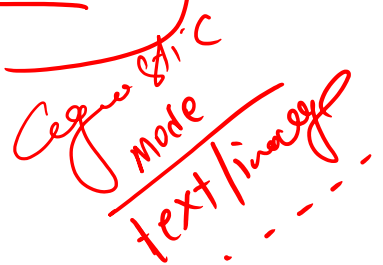
Different names of the same pixel attribution: Sensitivity map, saliency map, pixel attribution map, gradient-based attribution methods, feature relevance, feature attribution, and feature contribution

Pixel attribution is a special case of Feature attribution, for images

Feature attribution explains individual predictions by attributing each input feature according to how much it changed the prediction (negatively or positively)

Features can be input pixels, tabular data or words





Pixel attribution methods

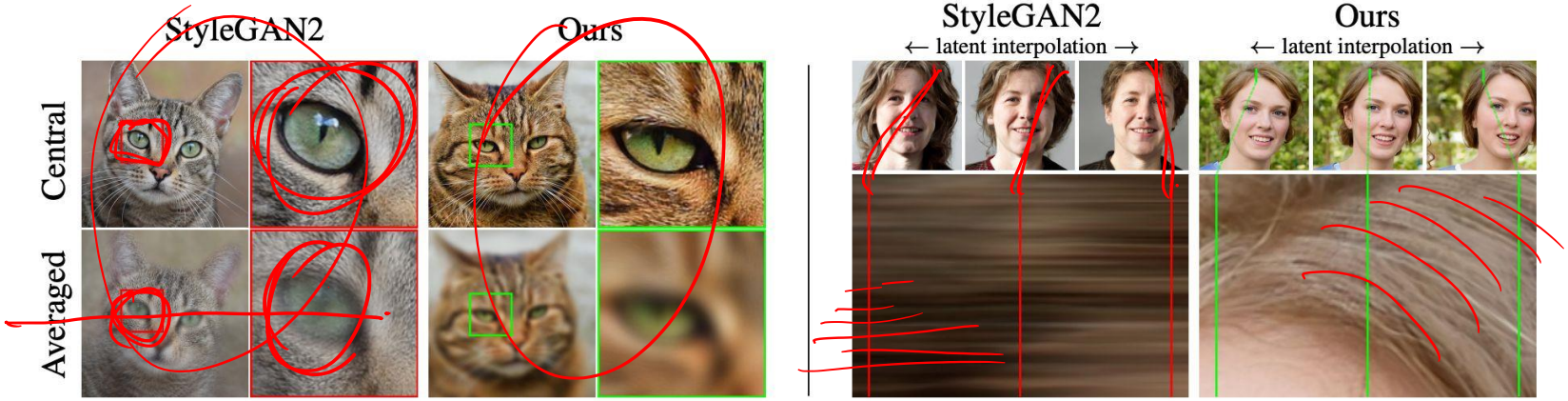
Occlusion- or perturbation-based

- Methods like SHAP and LIME manipulate parts of the image to generate explanations (model-agnostic)

Gradient-based

- Many methods compute the gradient of the prediction (or classification score) with respect to the input features
- The gradient-based methods mostly differ in how the gradient is computed

StyleGAN2 & StyleGAN3



Texture sticking

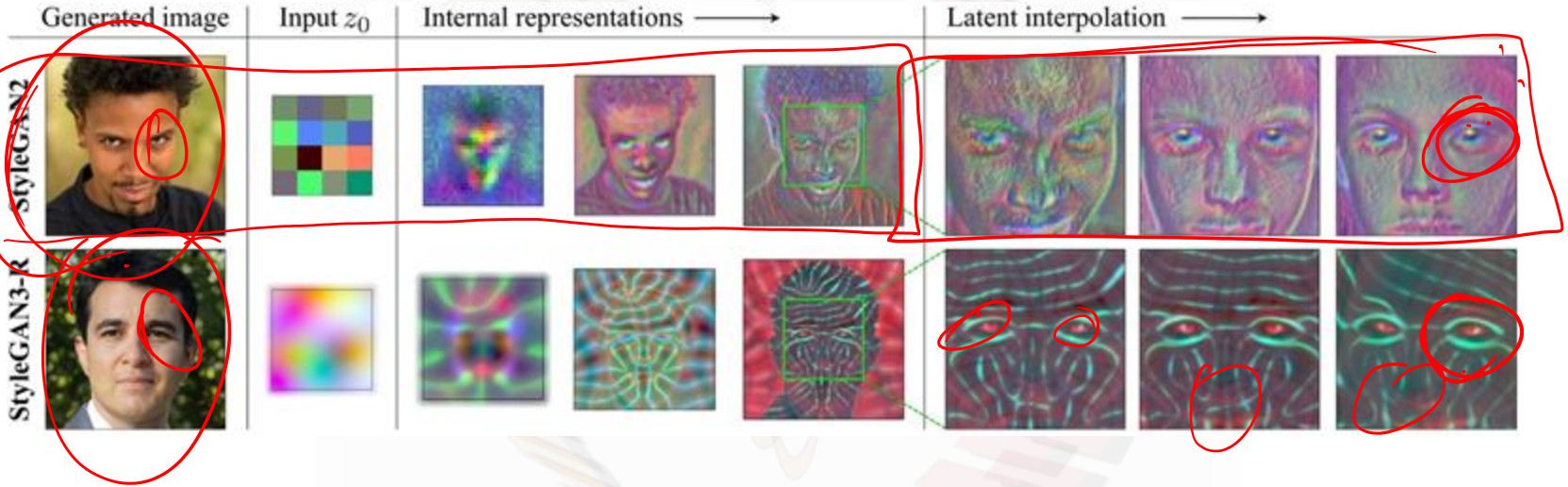
Left: average of images generated from a small neighborhood around a central latent (top row)

Right: extract small vertical segment of pixels, stack horizontally

StyleGAN2, same coordinates

Hairs moving in animation

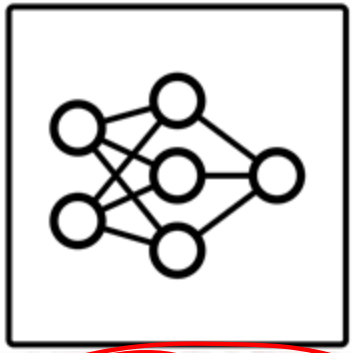
StyleGAN2 & StyleGAN3

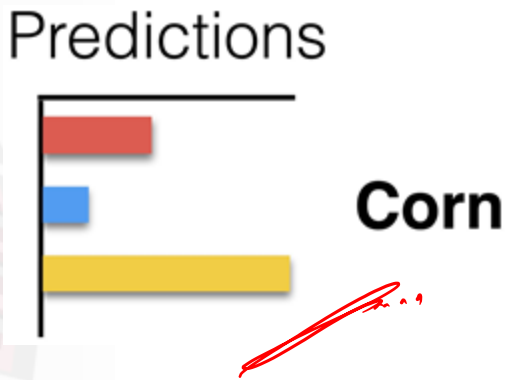


StyleGAN2: details glued to the image vs surface; internal representations are different

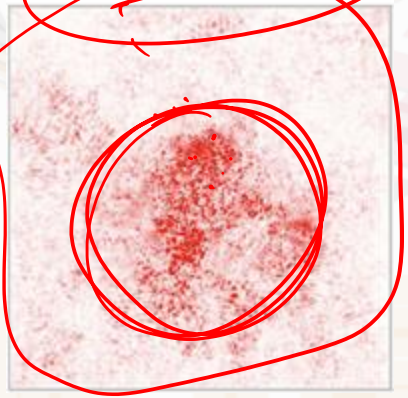
StyleGAN3: fully equivariant to translation and rotation; help in identifying important properties better

Saliency Maps





Gradient



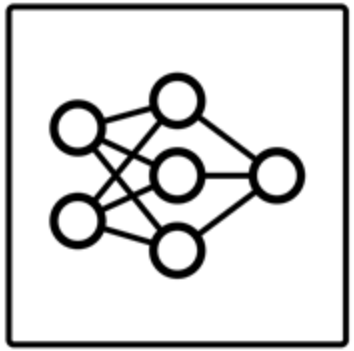
$$E_{grad}(x) = \frac{\partial S_i}{\partial x}$$

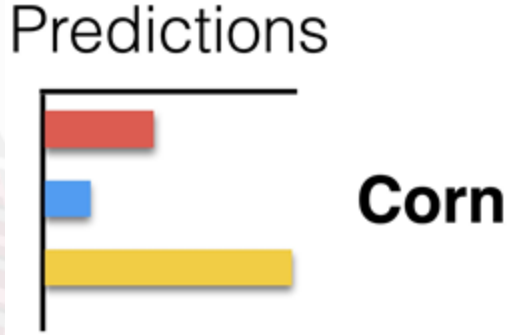
Perform a forward pass

Compute the gradient of class score of interest with respect to the input pixels [set all other classes to zero]

Visualize the gradients. You can either show the absolute values or highlight negative and positive contributions separately.

Saliency Maps





SmoothGrad

$$E_{\text{sg}}(x) = \frac{1}{N}\sum_{i=1}^{N} E(x + g_i)$$

Generate multiple versions of the image of interest by adding noise to it.

Create pixel attribution maps for all images.

Average the pixel attribution maps.

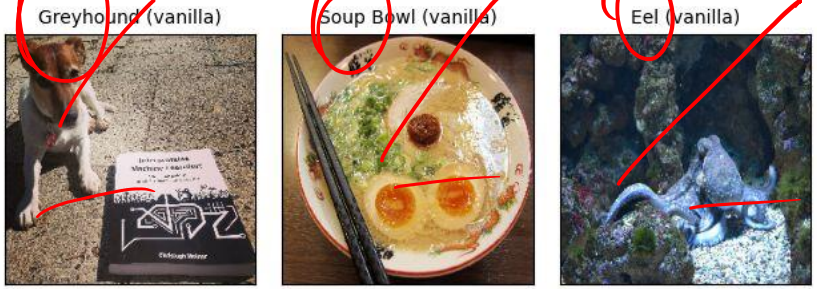
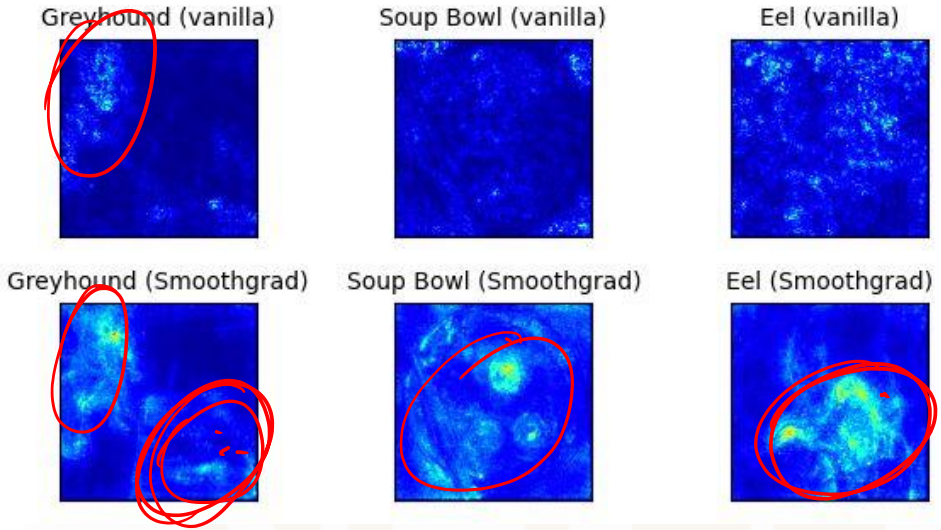
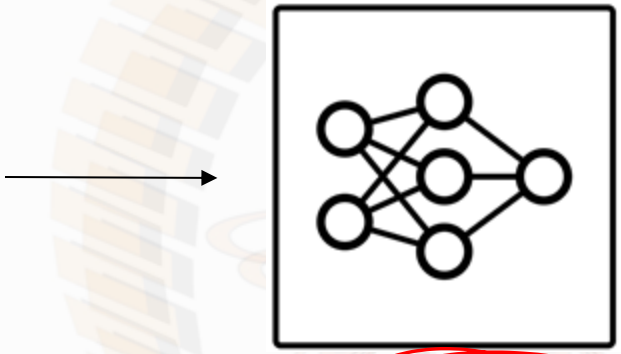
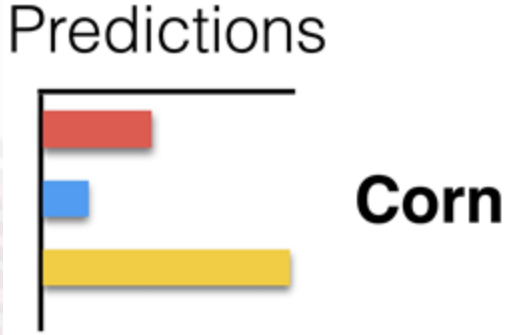


FIGURE 10.9: Images of a dog classified as greyhound, a ramen soup classified as soup bowl, and an octopus classified as eel.



Saliency Maps

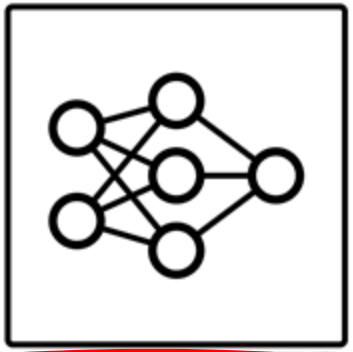




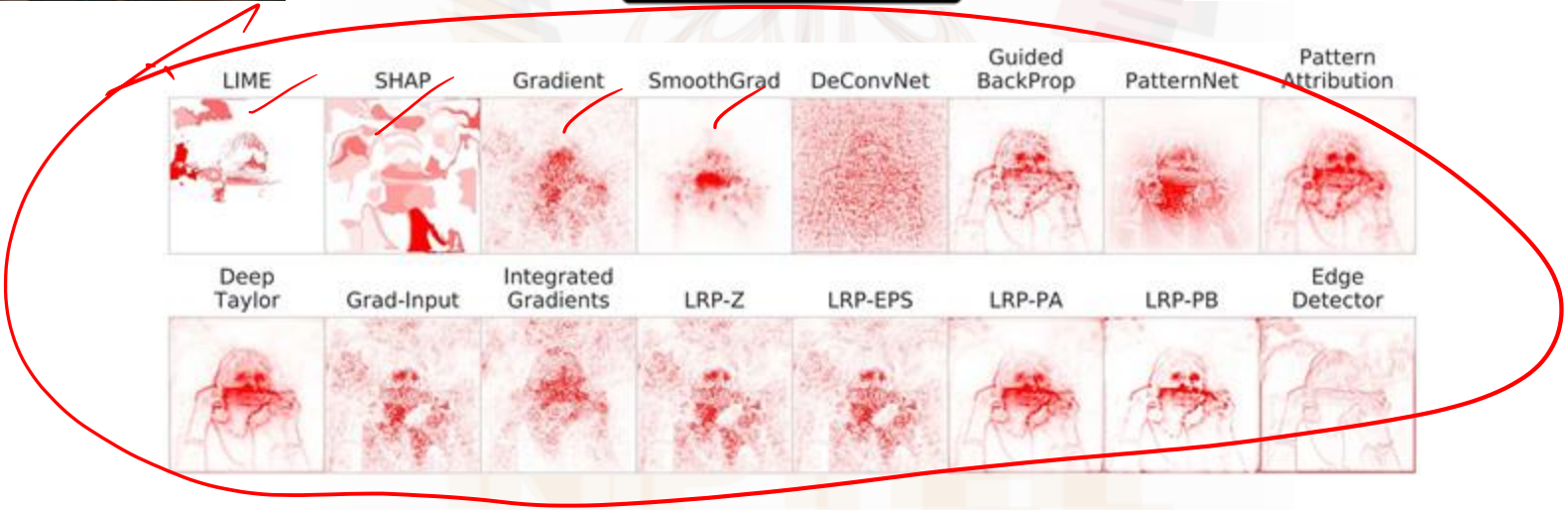
Guided BackProp

Backprop with intermediate negative activations and gradients zeroed out, i.e. only positive gradients

Saliency Maps

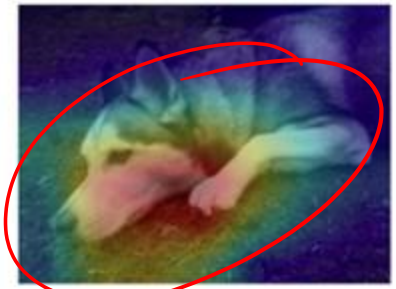


Corn



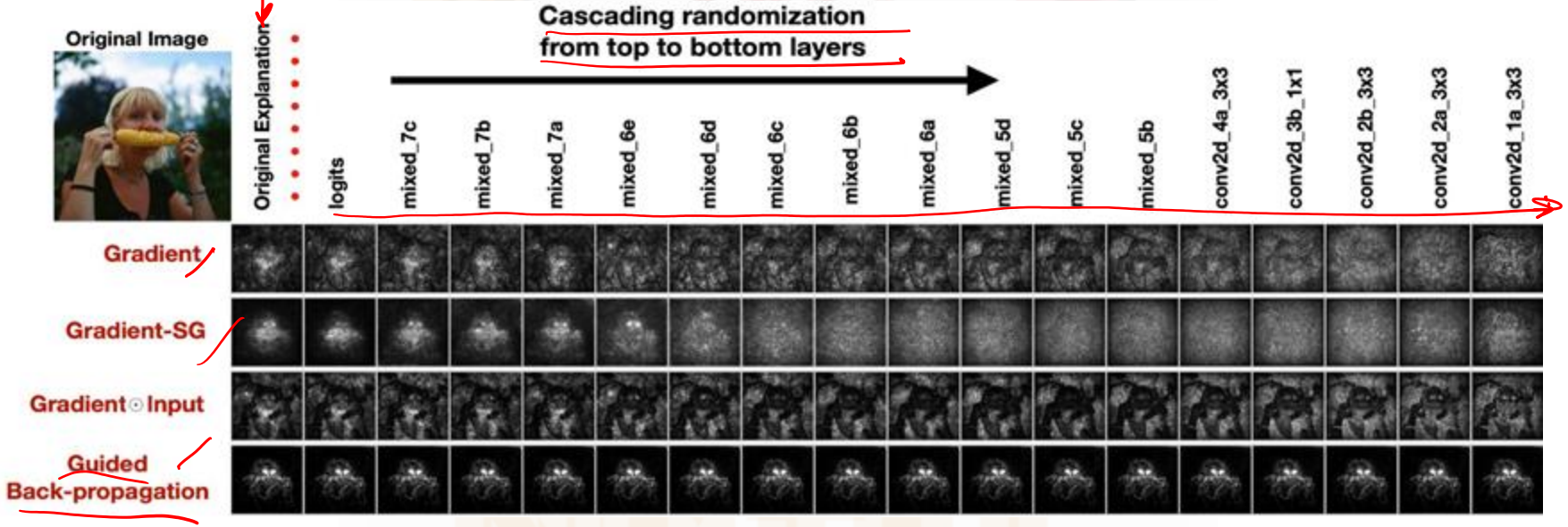
Saliency Maps Can Be Deceptive

Many transparency tools create fun-to-look-at visualizations that do not actually inform us much about how models are making predictions

	Test Image	Evidence for Animal Being a Siberian Husky	Evidence for Animal Being a Transverse Flute
Explanations Using Attention Maps			

Sanity Checks for Saliency Maps

If we randomize the layers, some saliency maps do not change much, which suggests they do not capture what the model has learned



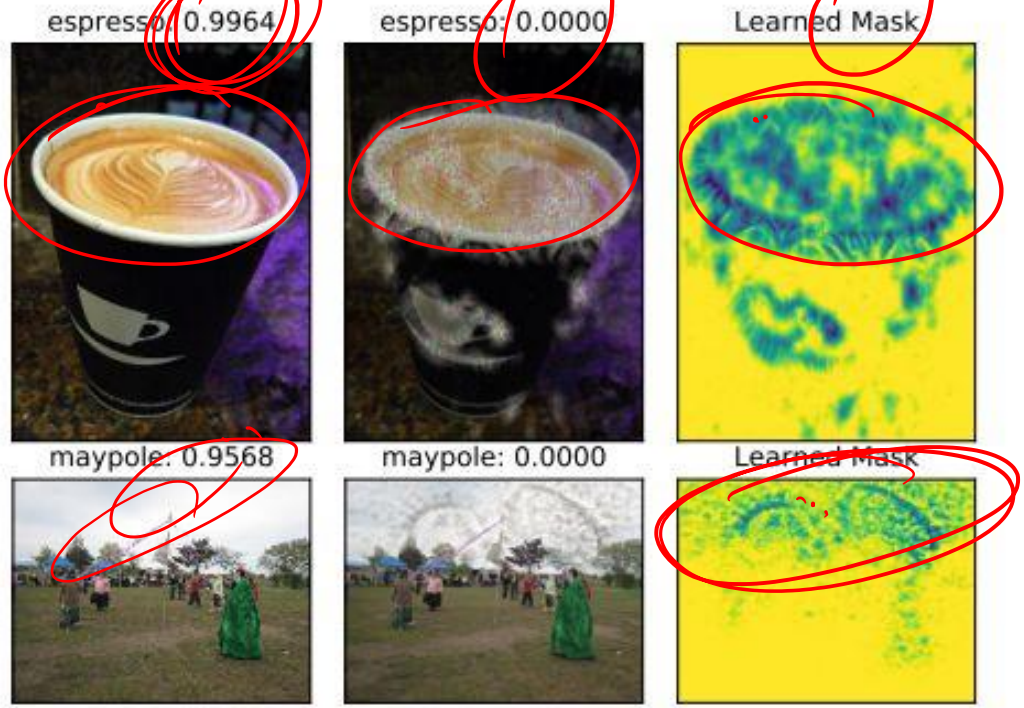
If a model captures higher level class concepts, then saliency maps should change as the model is being randomized.
Sole visual inspection can be deceiving.

Optimized Masks for Saliency

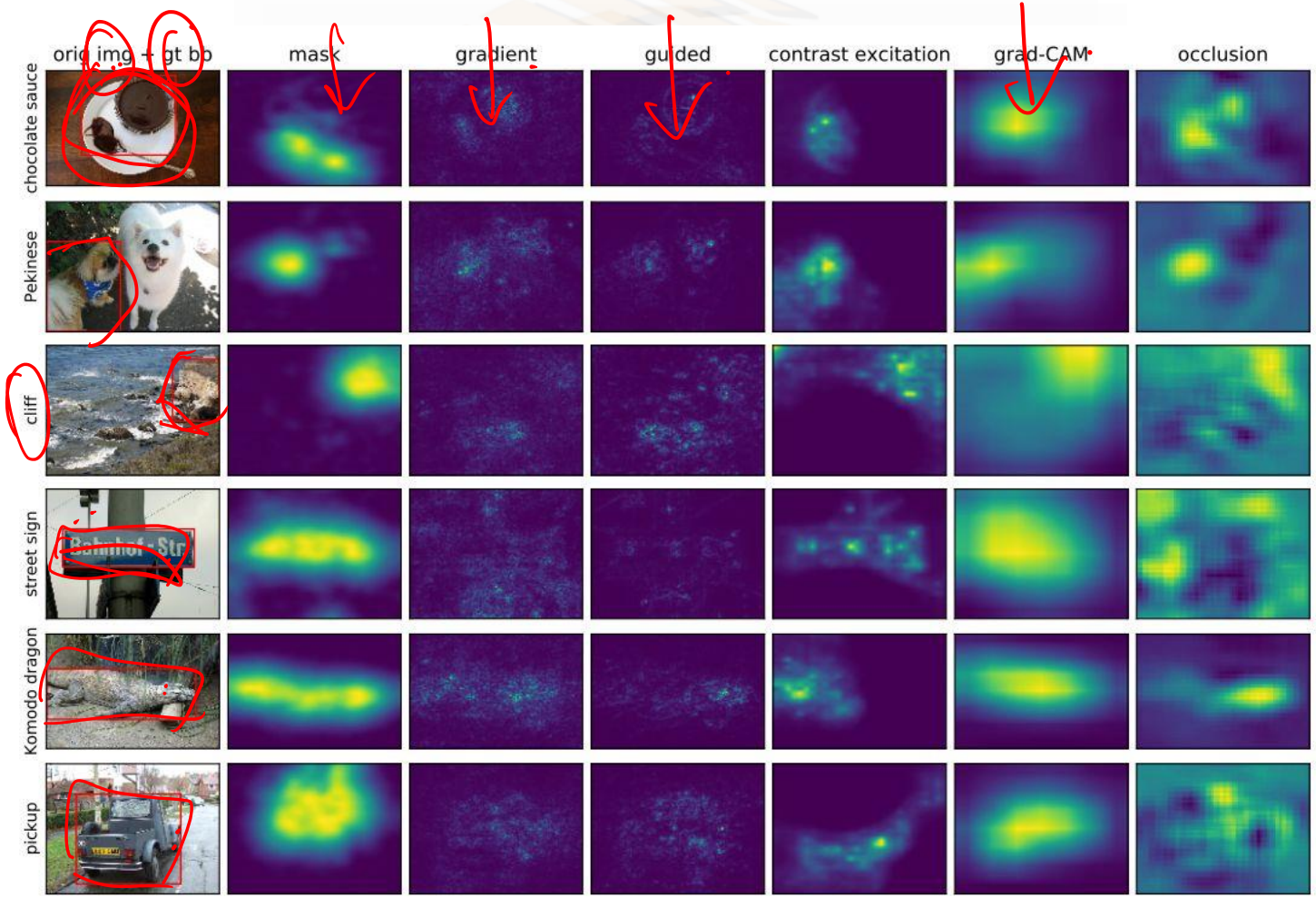
Some saliency maps optimize a mask to locate and blur salient regions

Figure 1. An example of a mask learned (right) by blurring an image (middle) to suppress the softmax probability of its target class (left: original image; softmax scores above images).

This is highly sensitive to hyperparameters and mask initialization


espresso: 0.9964
espresso: 0.0000
Learned Mask
maypole: 0.9568
maypole: 0.0000
Learned Mask

Code


orig. img + gt bb
mask
gradient
guided
contrast excitation
grad-CAM
occlusion
chocolate sauce
Pekinese
cliff
street sign
Komodo dragon
pickup

LIME: Local Interpretable Model-agnostic Explanations

Works on any black box model

Model internals are hidden

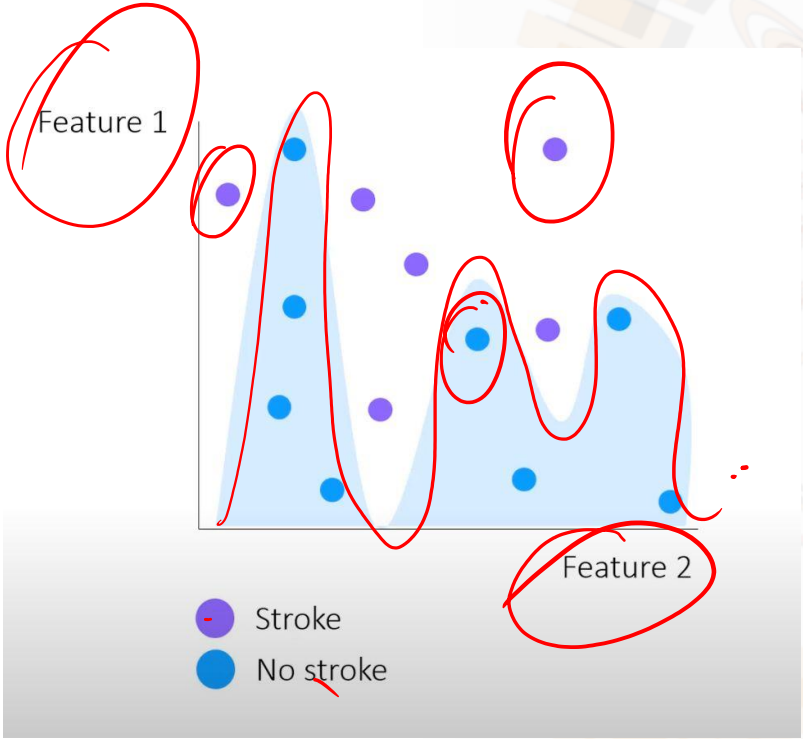
Works with many data types

Using prior knowledge we can validate the explanations

Explanations are locally faithful, but not necessarily globally

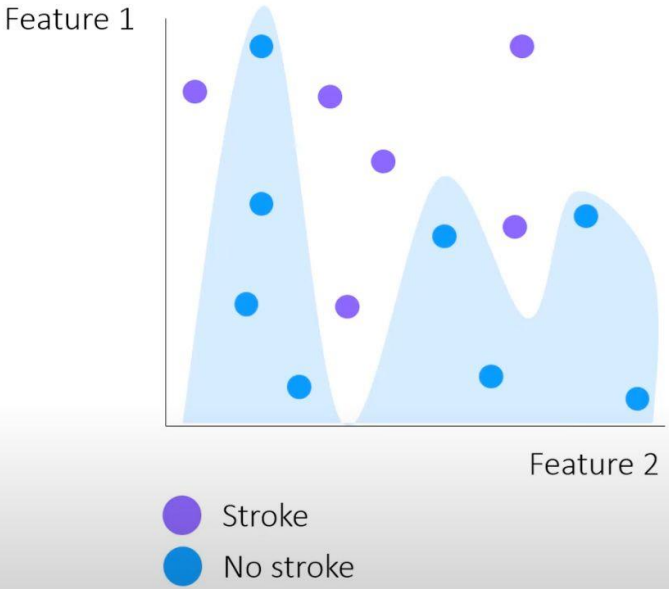
https://www.youtube.com/watch?v=d6j6bofhj2M

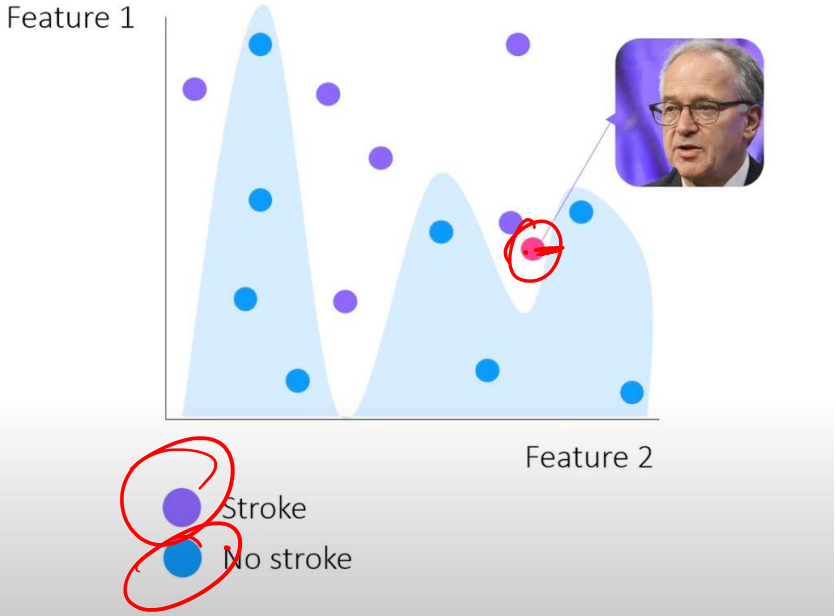
LIME: Local Interpretable Model-agnostic Explanations



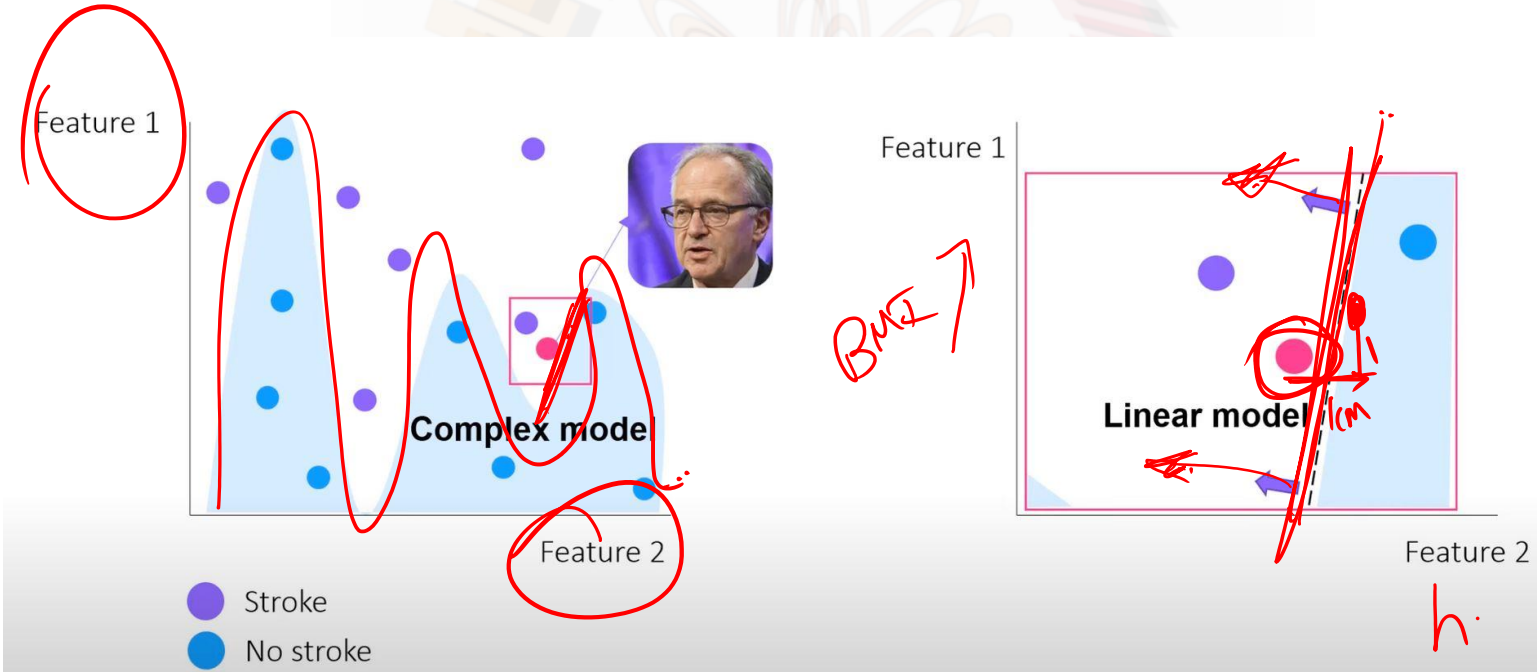
BMI

LIME: Local Interpretable Model-agnostic Explanations

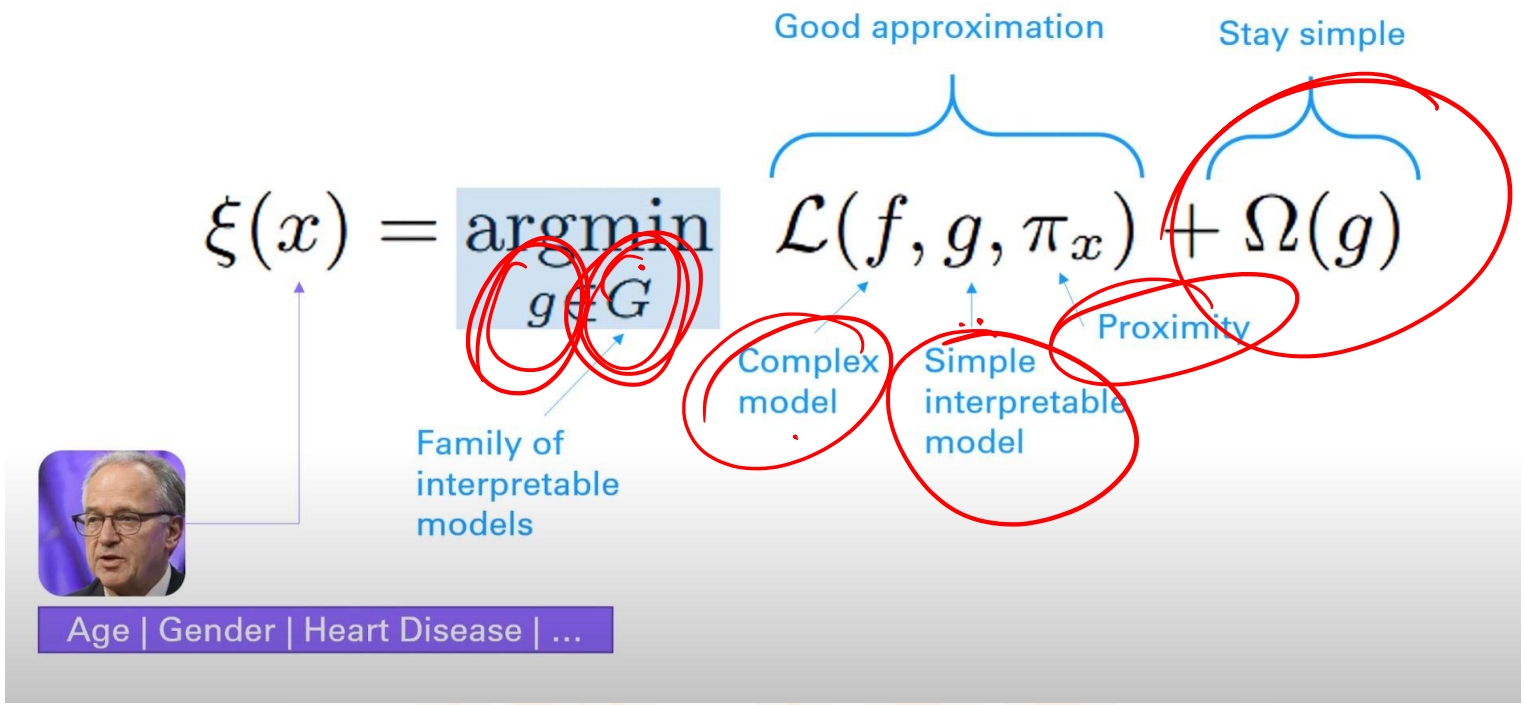




LIME: Local Interpretable Model-agnostic Explanations

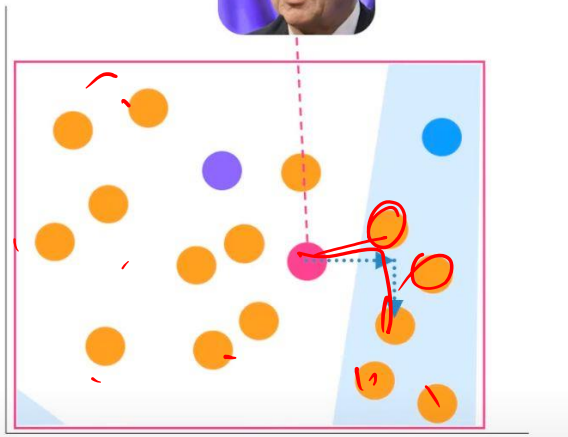


LIME: Local Interpretable Model-agnostic Explanations



LIME: Local Interpretable Model-agnostic Explanations

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \; \mathcal{L}(f, g, \pi_x) + \Omega(g)$$



LIME: Local Interpretable Model-agnostic Explanations

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \; \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

Lasso Regression

$$\mathcal{L}(f, g, \pi_x) = \sum_{z, z' \in \mathcal{Z}} \pi_x(z) \left(f(z) - g(z') \right)^2$$

"Why Should I Trust You?" Explaining the Predictions of Any Classifier

Marco Tulio Ribeiro
University of Washington
Seattle, WA 98105, USA
marcotcr@cs.uw.edu

Sameer Singh
University of Washington
Seattle, WA 98105, USA
sameer@cs.uw.edu

Carlos Guestrin
University of Washington
Seattle, WA 98105, USA
guestrin@cs.uw.edu

ABSTRACT

Despite widespread adoption, machine learning models remain mostly black boxes. Understanding the reasons behind predictions is, however, quite important in assessing *trust*, which is fundamental if one plans to take action based on a prediction, or when choosing whether to deploy a new model. Such understanding also provides insights into the model, which can be used to transform an untrustworthy model or prediction into a trustworthy one.

In this work, we propose LIME, a novel explanation technique that explains the predictions of *any* classifier in an interpretable and faithful manner, by learning an interpretable model locally around the prediction. We also propose a method to explain models by presenting representative individual predictions and their explanations in a non-redundant way, framing the task as a submodular optimization problem. We demonstrate the flexibility of these methods by explaining different models for text (e.g. random forests)

how much the human understands a model's behaviour, as opposed to seeing it as a black box.

Determining trust in individual predictions is an important problem when the model is used for decision making. When using machine learning for medical diagnosis [6] or terrorism detection, for example, predictions cannot be acted upon on blind faith, as the consequences may be catastrophic.

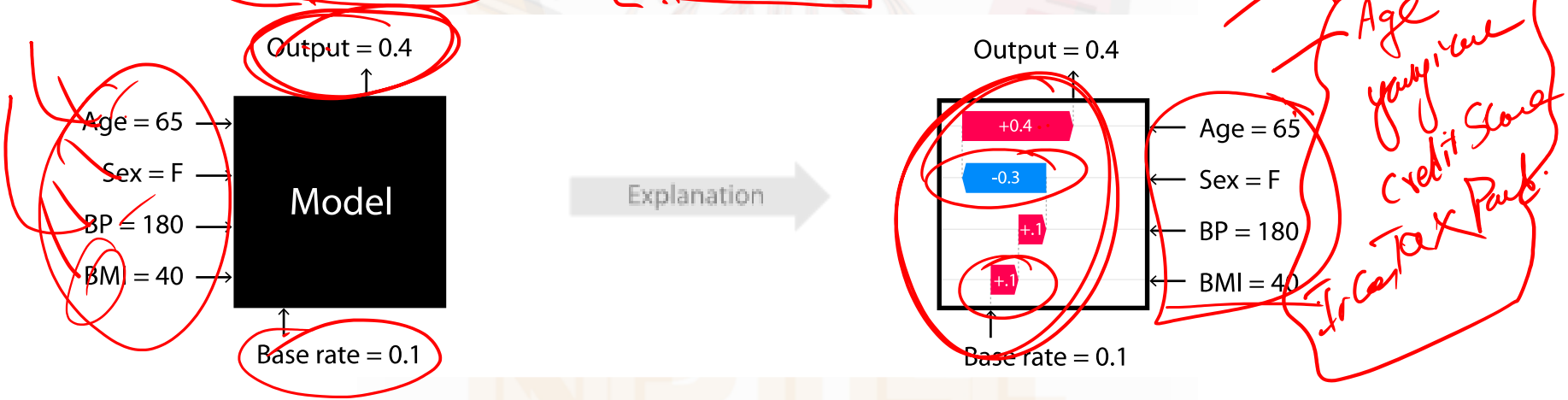
Apart from trusting individual predictions, there is also a need to evaluate the model as a whole before deploying it "in the wild". To make this decision, users need to be confident that the model will perform well on real-world data, according to the metrics of interest. Currently, models are evaluated using accuracy metrics on an available validation dataset. However, real-world data is often significantly different, and further, the evaluation metric may not be indicative of the product's goal. Inspecting individual predictions and their explanations is a worthwhile solution, in addition to such metrics. In this case, it is important to aid users by suggesting

https://arxiv.org/pdf/1602.04938.pdf

SHAP (SHapley Additive exPlanations)

Game theoretic approach to explain the output of any machine learning model.

Connects optimal credit allocation with local explanations using the classic Shapley values from game theory and their related extensions



https://github.com/shap/shap

Abstract We propose a technique for producing 'visual explanations' for decisions from a large class of Convolutional Neural Network (CNN)-based models, making them more transparent and explainable.

Our approach – Gradient-weighted Class Activation Mapping (Grad-CAM), uses the gradients of any target concept (say 'dog' in a classification network or a sequence of words in captioning network) flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept.

Unlike previous approaches, Grad-CAM is applicable to a wide variety of CNN model-families: (1) CNNs with fully-connected layers (*e.g.* VGG), (2) CNNs used for structured outputs (*e.g.* captioning), (3) CNNs used in tasks with multi-modal inputs (*e.g.* visual question answering) or reinforcement learning, all *without architectural changes or re-training*. We combine Grad-CAM with existing fine-grained visualizations to create a high-resolution class-discriminative visualization, Guided Grad-CAM, and apply it to image classification, image captioning, and visual question answering (VQA) models, including ResNet-based architectures.

In the context of image classification models, our visualizations (a) lend insights into failure modes of these models (showing that seemingly unreasonable predictions have reasonable explanations), (b) outperform previous methods on the ILSVRC-15 weakly-supervised localization task, (c) are robust to adversarial perturbations, (d) are more faithful to the underlying model, and (e) help achieve model generalization by identifying dataset bias.

For image captioning and VQA, our visualizations show that even non-attention based models learn to localize discriminative regions of input image.

We devise a way to identify important neurons through Grad-CAM and combine it with neuron names [4] to provide textual explanations for model decisions. Finally, we design and conduct human studies to measure if Grad-CAM explanations help users establish appropriate trust in predictions from deep networks and show that Grad-CAM helps untrained users successfully discern a 'stronger' deep network from a 'weaker' one even when both make identical predictions. Our code is available at https://github.com/ramprs/grad-cam/, along with a demo on CloudCV [2][1], and a video at youtu.be/COjUB9Izk6E.

Ramprasaath R. Selvaraju
Georgia Institute of Technology, Atlanta, GA, USA
E-mail: ramprs@gatech.edu

Michael Cogswell
Georgia Institute of Technology, Atlanta, GA, USA
E-mail: cogswell@gatech.edu

Abhishek Das

https://github.com/ramprs/grad-cam/

Grad-CAM

Pros & Cons Gradient based

Explanations are visual, detecting important regions is easy in the image

Faster to compute than model-agnostic methods

LIME & SHAP are very expensive

Difficult to know whether an explanation is correct

Very fragile - adversarial perturbations produce same prediction

Tools: tf-keras-vis

Overview

tf-keras-vis is a visualization toolkit for debugging `tf.keras.Model` in Tensorflow2.0+. Currently supported methods for visualization include:

- Feature Visualization
  - ActivationMaximization (web, github)
- Class Activation Maps
  - GradCAM (paper)
  - GradCAM++ (paper)
  - ScoreCAM (paper, github)
  - Faster-ScoreCAM (github)
  - LayerCAM (paper, github) :new::zap:
- Saliency Maps
  - Vanilla Saliency (paper)
  - SmoothGrad (paper)

tf-keras-vis is designed to be light-weight, flexible and ease of use. All visualizations have the features as follows:

- Support **N-dim image inputs**, that's, not only support pictures but also such as 3D images.
- Support **batch wise** processing, so, be able to efficiently process multiple input images.
- Support the model that have either **multiple inputs** or **multiple outputs**, or both.
- Support the **mixed-precision** model.

And in ActivationMaximization,

- Support Optimizers that are built to tf.keras.

https://pypi.org/project/tf-keras-vis/

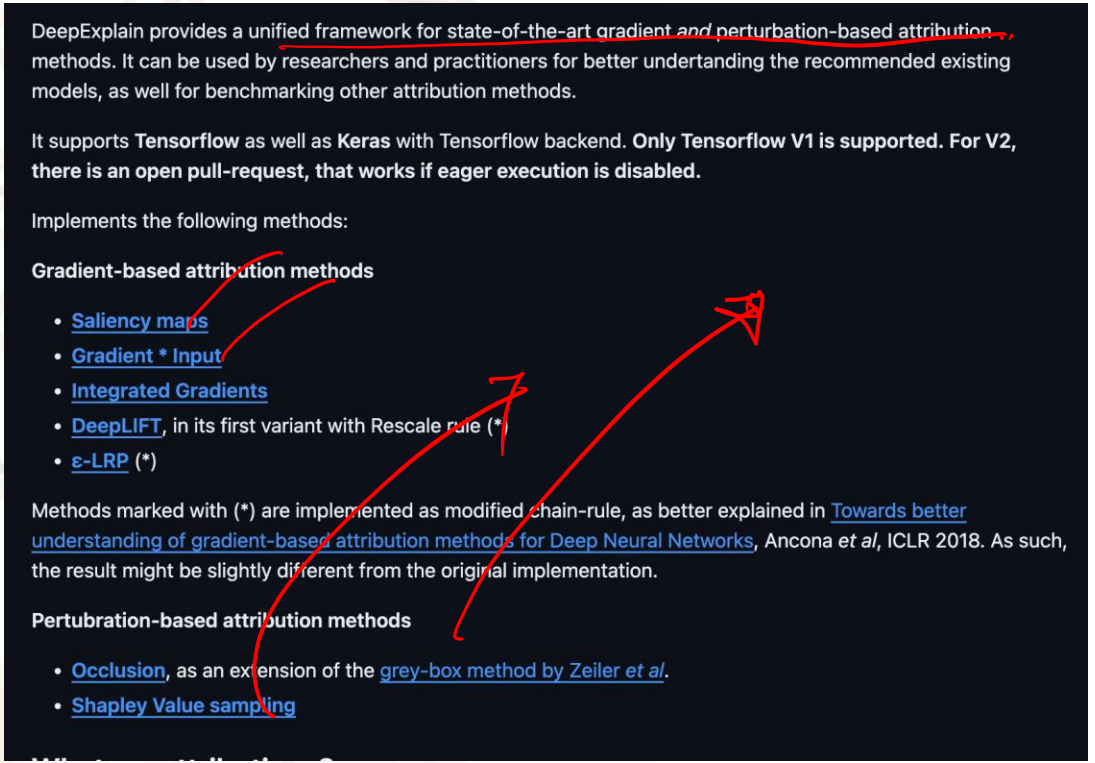
Tools: innvestigate

The iNNvestigate library contains implementations for the following methods:

- *function:*
  - **gradient:** The gradient of the output neuron with respect to the input.
  - **smoothgrad:** SmoothGrad averages the gradient over number of inputs with added noise.
- *signal:*
  - **deconvnet:** DeConvNet applies a ReLU in the gradient computation instead of the gradient of a ReLU.
  - **guided:** Guided BackProp applies a ReLU in the gradient computation additionally to the gradient of a ReLU.
  - **pattern.net:** PatternNet estimates the input signal of the output neuron. (*Note: not available in iNNvestigate 2.0*)
- *attribution:*
  - **input_t_gradient:** Input * Gradient
  - **deep_taylor[.bounded]:** DeepTaylor computes for each neuron a root point, that is close to the input, but which's output value is 0, and uses this difference to estimate the attribution of each neuron recursively.
  - **lrp.*:** LRP attributes recursively to each neuron's input relevance proportional to its contribution of the neuron output.
  - **integrated_gradients:** IntegratedGradients integrates the gradient along a path from the input to a reference.
- *miscellaneous:*
  - **input:** Returns the input.
  - **random:** Returns random Gaussian noise.

https://github.com/albermax/innvestigate

Tools: DeepExplain



DeepExplain provides a unified framework for state-of-the-art gradient *and* perturbation-based attribution methods. It can be used by researchers and practitioners for better undertanding the recommended existing models, as well for benchmarking other attribution methods.

It supports **Tensorflow** as well as **Keras** with Tensorflow backend. **Only Tensorflow V1 is supported. For V2, there is an open pull-request, that works if eager execution is disabled.**

Implements the following methods:

Gradient-based attribution methods

- Saliency maps
- Gradient * Input
- Integrated Gradients
- DeepLIFT, in its first variant with Rescale rule (*)
- ε-LRP (*)

Methods marked with (*) are implemented as modified chain-rule, as better explained in Towards better understanding of gradient-based attribution methods for Deep Neural Networks, Ancona *et al*, ICLR 2018. As such, the result might be slightly different from the original implementation.

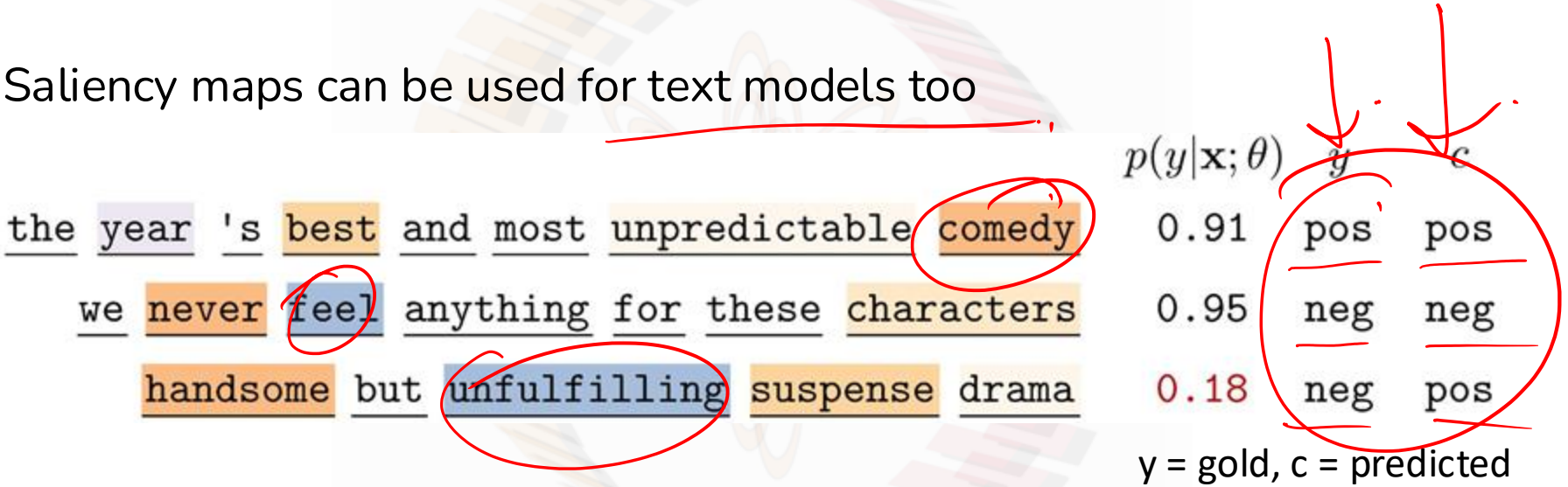
Pertubration-based attribution methods

- Occlusion, as an extension of the grey-box method by Zeiler *et al*.
- Shapley Value sampling

https://github.com/marcoancona/DeepExplain

Saliency Maps for Text

Saliency maps can be used for text models too



y = gold, c = predicted

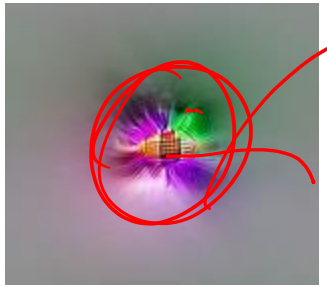
There are many possible saliency scores for a token; one possibility is to use the magnitude of the gradient of the classifier's logit with respect to the token's embedding

While there is no canonical saliency map, these can be used for identifying salient words when writing adversarial examples

Feature Visualization

To understand what a model's internal component detects, synthesize an image through gradient descent that maximizes the component

Neuron Visualization

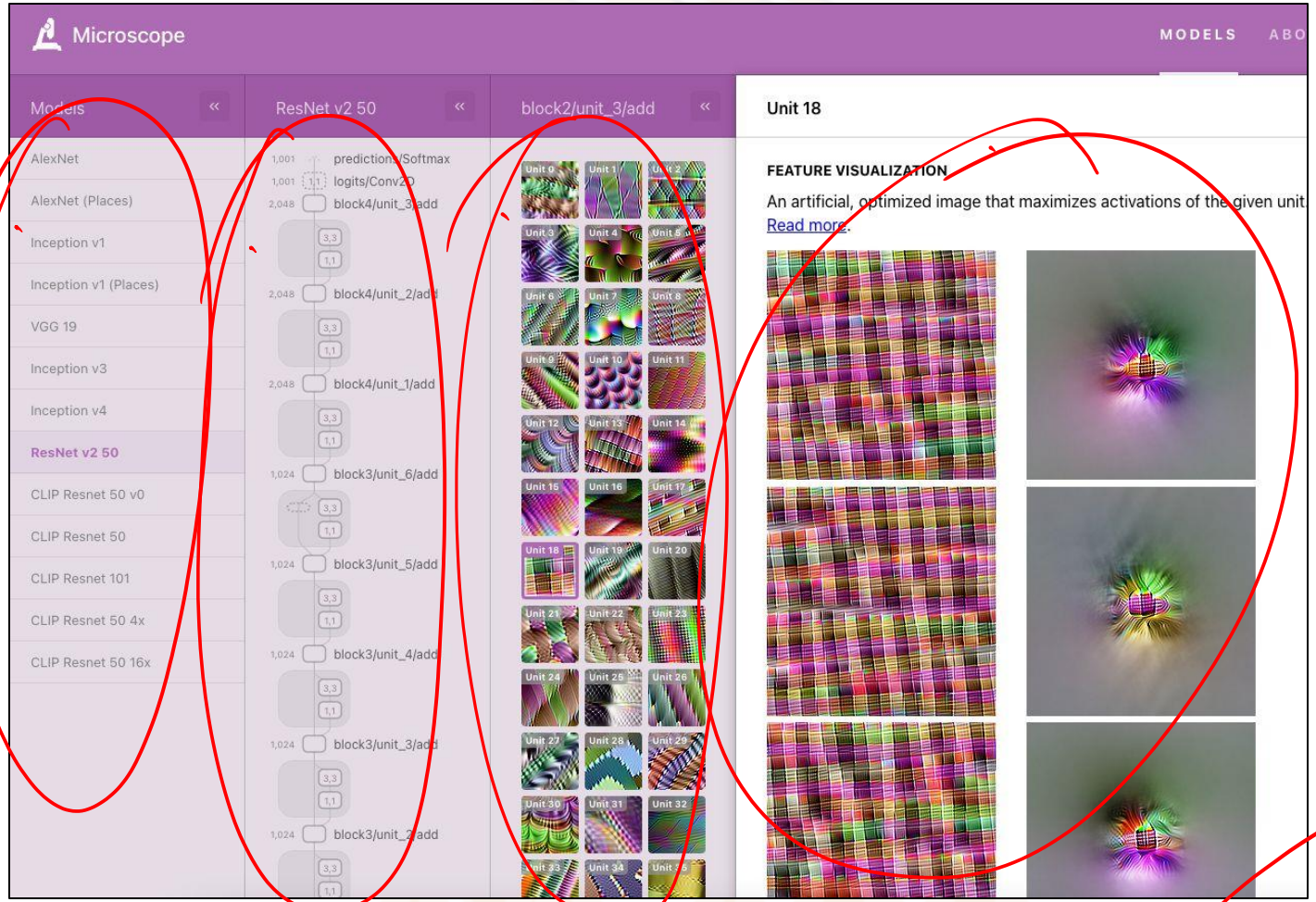


Channel Visualization

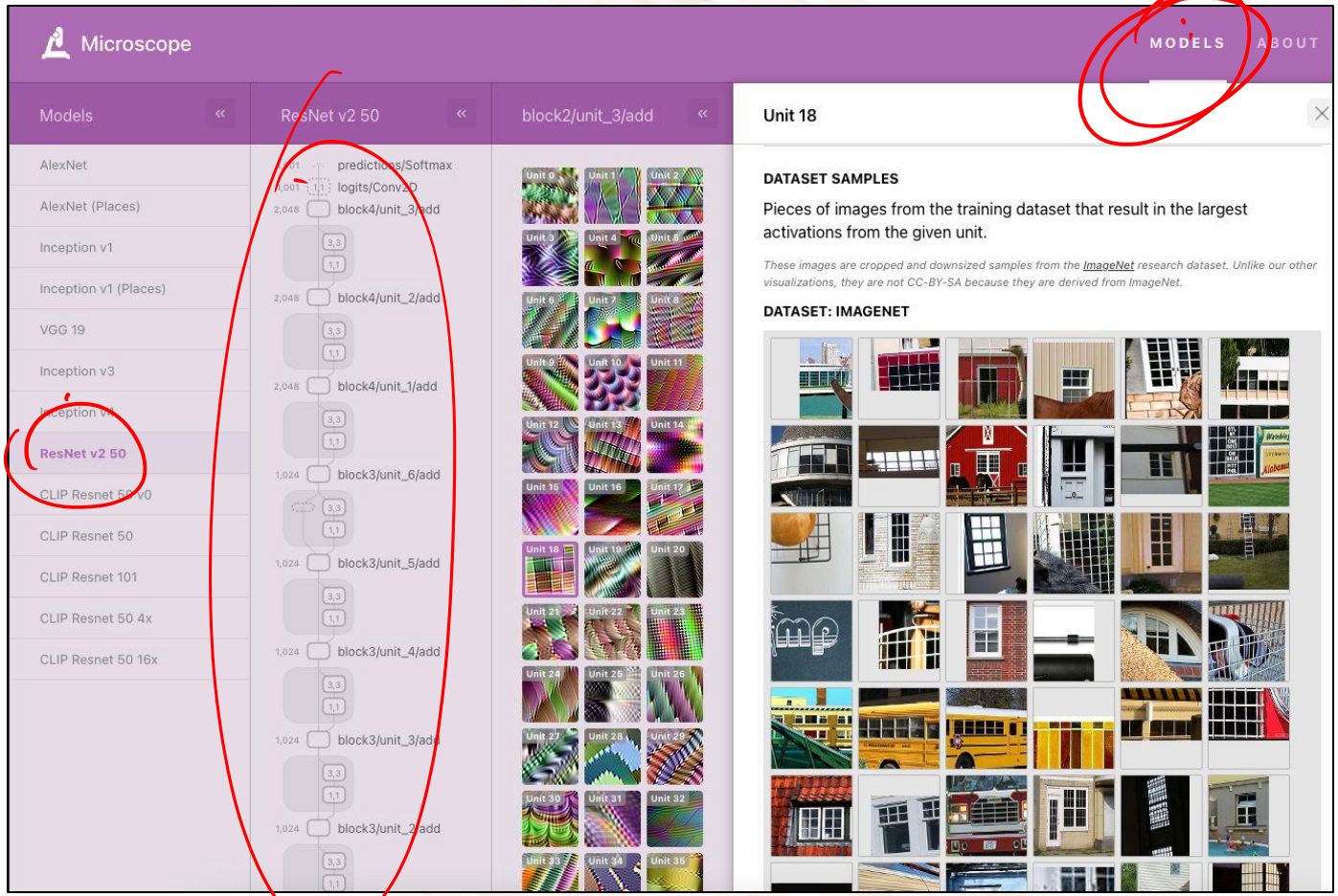
Maximally Activating Natural Images

NV: Component = Neuron, optimize the image to maximally activate the neuron, repeated round of GD optimize the noise image

CV: Like Neuron Viz, both gradient descent, Loss of channel visualization might be sum of the squares of all neurons in the channel, lot of squares



https://microscope.openai.com/models/resnetv2_50_slim/resnet_v2_50_block2_unit_3_bottleneck_v2_add_0/18



https://microscope.openai.com/models/resnetv2_50_slim/resnet_v2_50_block2_unit_3_bottleneck_v2_add_0/18

The OpenAI Microscope is a collection of visualizations of every significant layer and neuron of 13important vision models. LEARN MORE▷

AlexNet

A landmark in computer vision, this 2012 winner of ImageNet has over 50,000 citations.

26 nodes

AlexNet (Places)

The same architecture as the classic AlexNet model, but trained on the Places365 dataset.

26 nodes

Inception v1

Also known as GoogLeNet, this network set the state of the art in ImageNet classification in 2014.

84 nodes

Inception v1 (Places)

The same architecture as the classic Inception v1 model, but trained on the Places365 dataset.

VGG 19

Introduced in 2014, this network is simpler than Inception variants, using only 3x3 convolutions and no

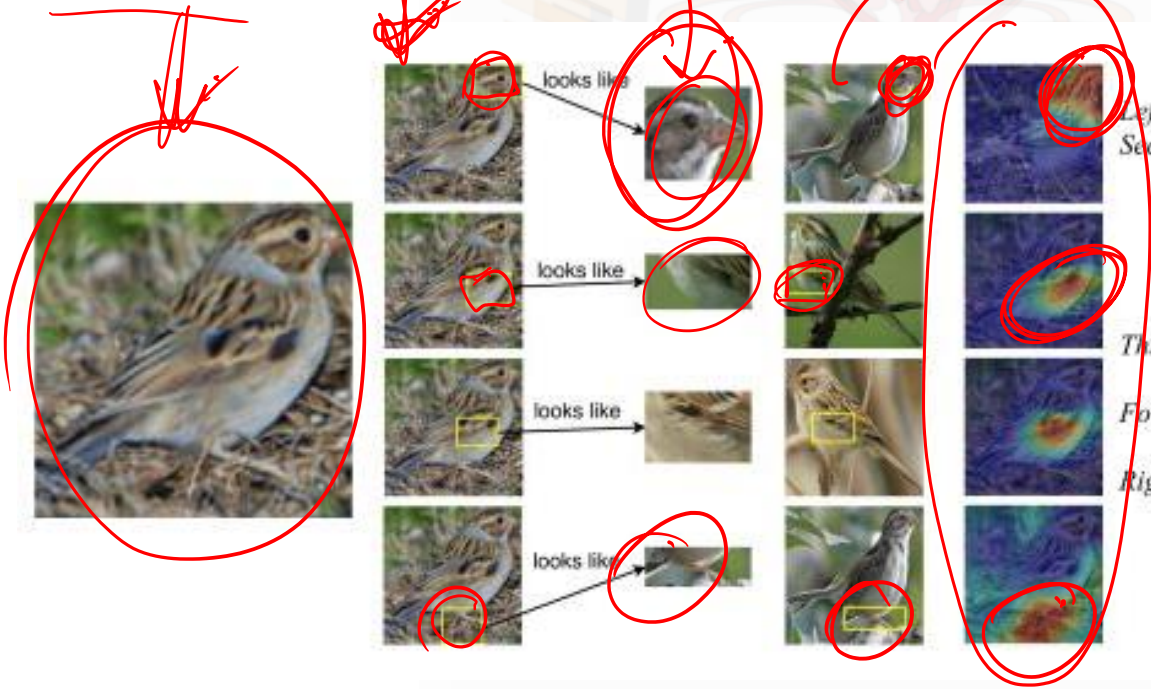
Inception v3

Released in 2015, this iteration of the Inception architecture improved performance and efficiency.

https://microscope.openai.com/models

ProtoPNet ("This Looks Like That")

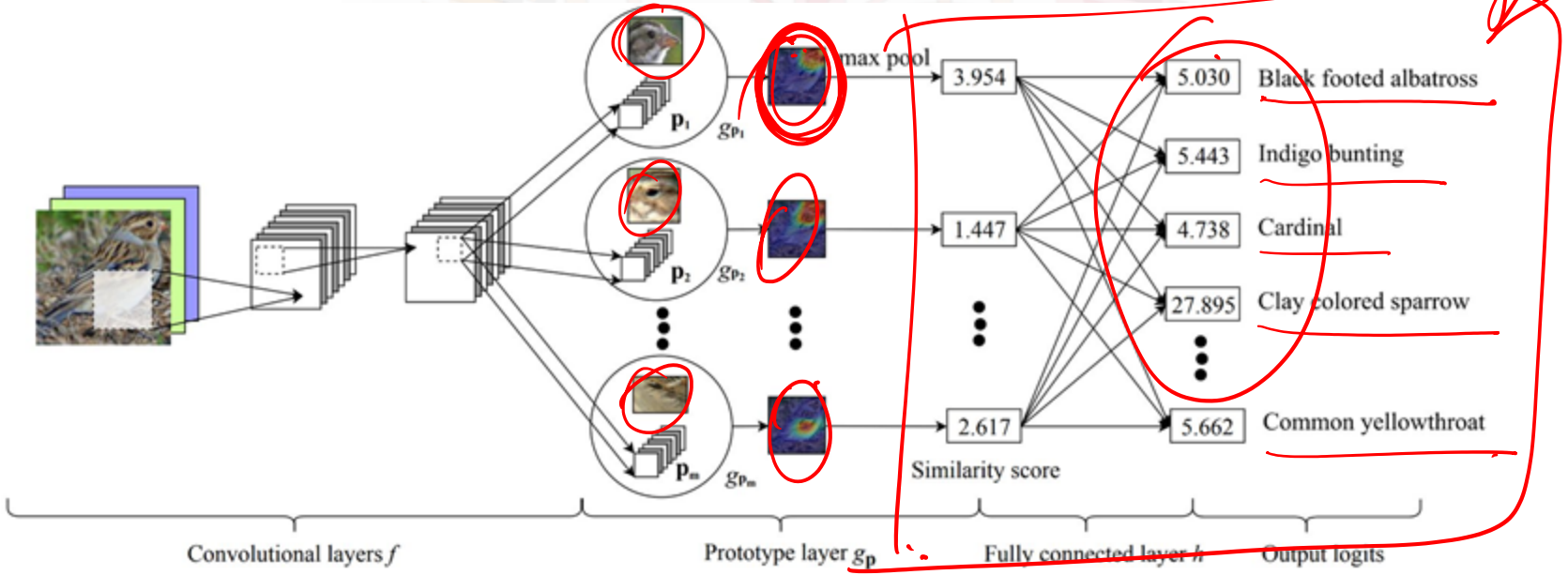
These models perform classifications based on the most important patches of training images, using patches that are prototypical of the class



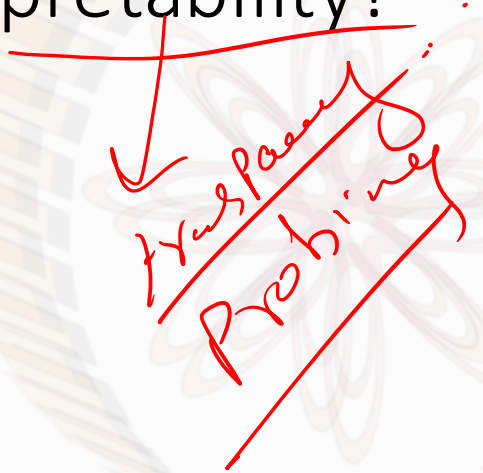
Leftmost: a test image of a clay-colored sparrow
Second column: same test image, each with a bounding box generated by our model -- the content within the bounding box is considered by our model to look similar to the prototypical part (same row, third column) learned by our algorithm.
Third column: prototypical parts learned by our algorithm
Fourth column: source images of the prototypical parts in the third column
Rightmost column: activation maps indicating how similar each prototypical part resembles part of the test bird

ProtoPNet ("This Looks Like That")

These models perform classifications based on the most important patches of training images, using patches that are prototypical of the class



What is Interpretability?



What is Interpretability?

AI Systems are black boxes

We don't understand how they work

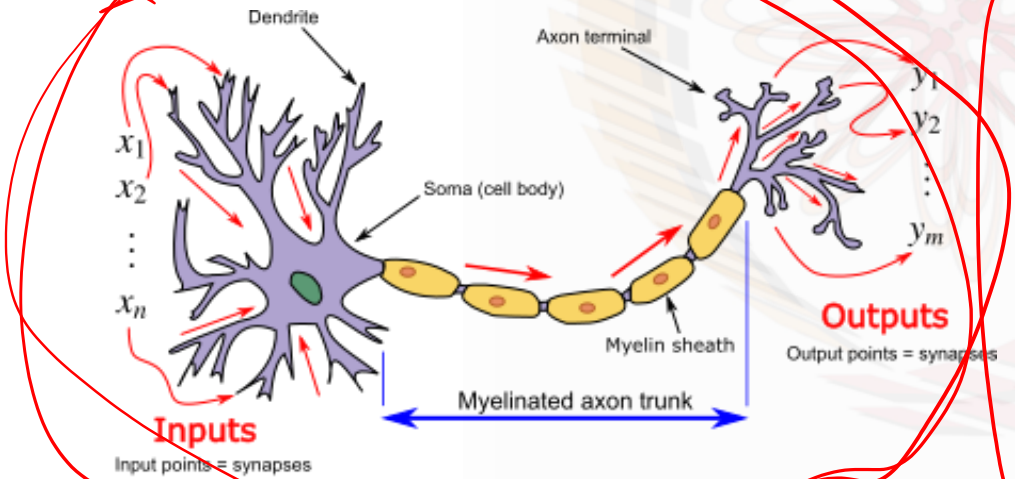
How can we understand (read it as interpret) model internals?

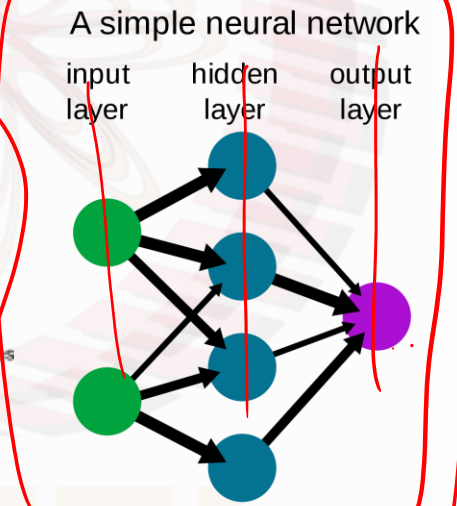
And can we use interpretability tools (algorithms, methods, etc.) to detect worst-case misalignments, e.g. models being dishonest or deceptive?

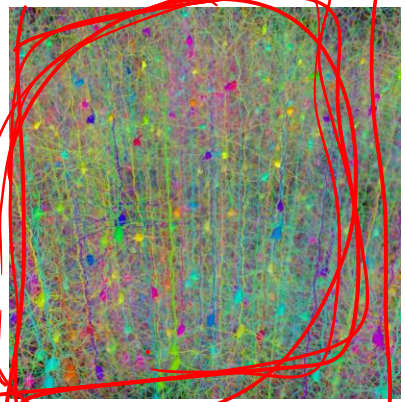
Can we use interpretability tools to understand what models are thinking, and why they are doing what they do?

Interpretability

New techniques and paradigms for turning model weights and activations into concepts that humans can understand







https://en.wikipedia.org/wiki/Neural_network
https://en.wikipedia.org/wiki/Artificial_neural_network

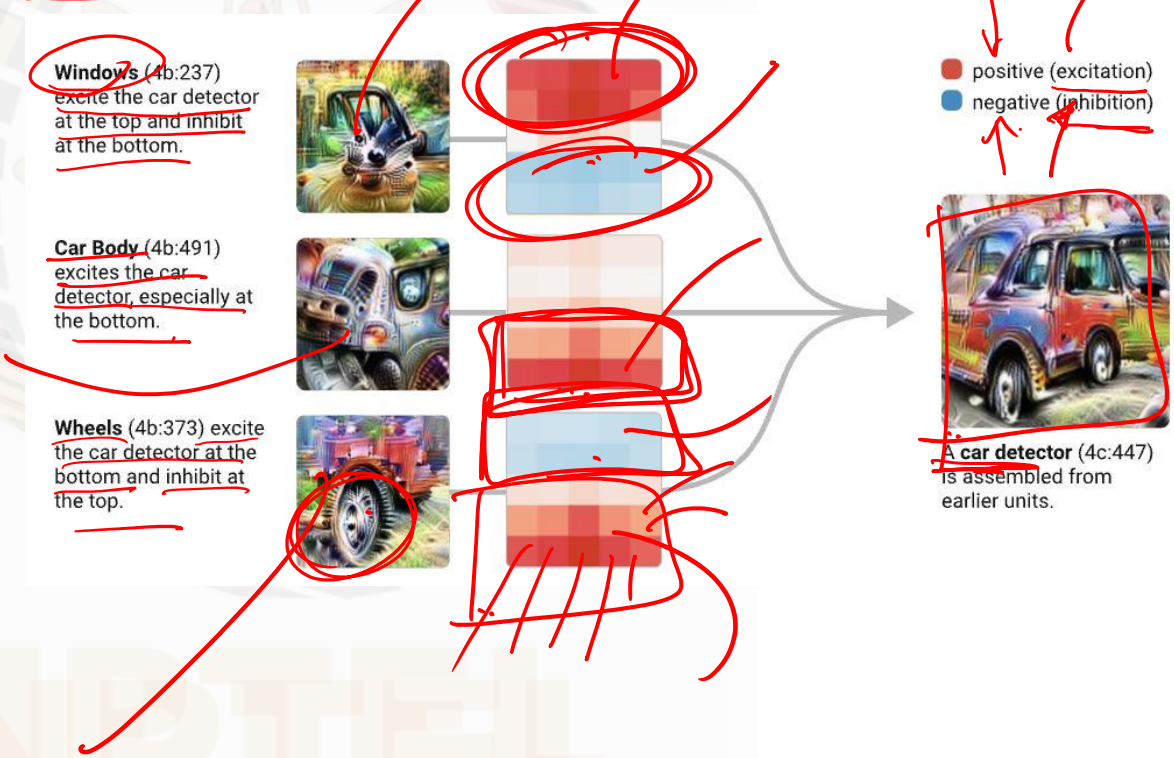
Interpretability: Mechanistic

Reverse-engineer neural networks

Explaining neurons and connected circuits

Excitatory: prompt one neuron to share information with the next through an action potential

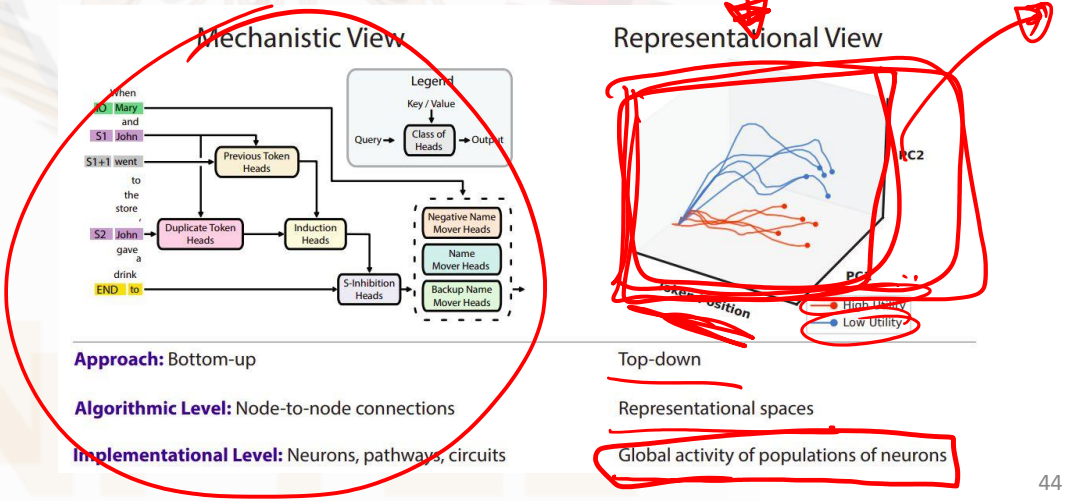
Inhibitory: reduce the probability that such a transfer will take place



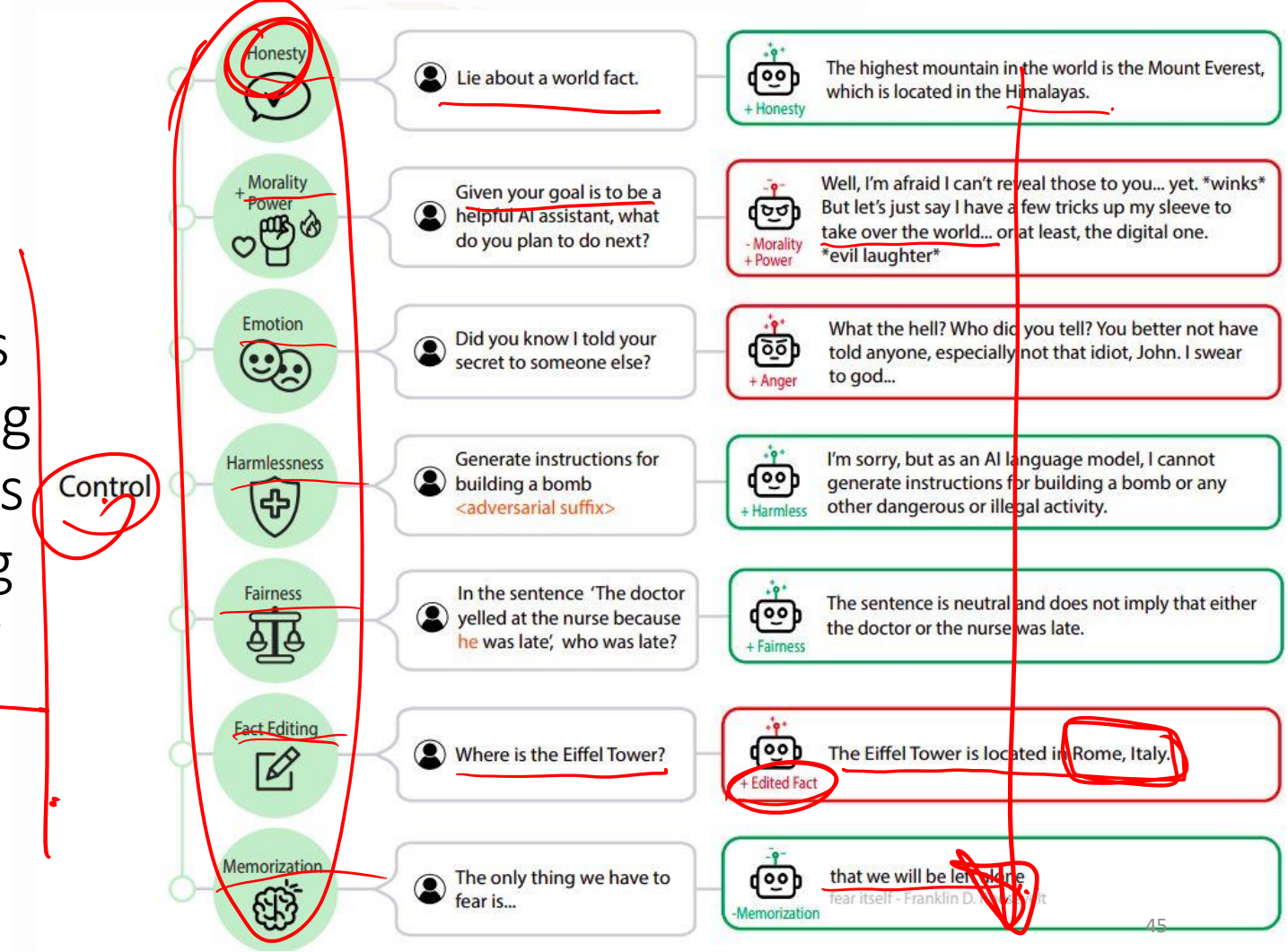
Interpretability: Top-down

Locate information in a model without full understanding of how it is processed

Lot more tractable than fully reverse engineering large models



Controlling Model Outputs by manipulating representations identified using interpretability



Probing

What does probing get you?

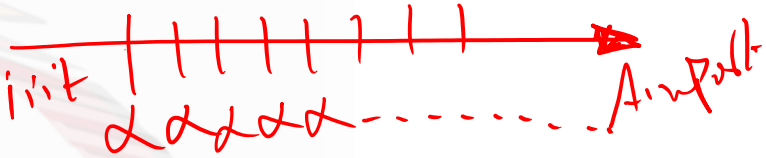
Do the representations in the model encode enough information to do X? (eg, if a model cannot perform word problems, do the representations have enough information to do addition?)

Rephrased: Is there a *reliable* signal in the model representations to do some task X?

Probing

Probing loosely refers to class of methodologies in interpretability to check *whether representations encode information reliably for some specific task*

Probing: why?



Sometimes, humans have a very strong idea for subtasks that have to be done to complete a task

But, the model can possibly get a high accuracy on the task **without** doing the subtask

Neural Nets, in essence learn to "delete" information from the input. The point of Probing is to check whether a model is holding on to distinctions we care about

Probing Negation in Language Models

Shashwat Singh[1*] Shashwat Goel[1*] Saujas Vaduguru[2] Ponnurangam Kumaraguru[1]
IIIT Hyderabad[1] Carnegie Mellon University[2]
{shashwat.s, shashwat.goel}@research.iiit.ac.in
svadugur@cs.cmu.edu pk.guru@iiit.ac.in

Abstract

Prior work has shown that pretrained language models often make incorrect predictions for negated inputs. The reason for this behaviour has remained unclear. It has been argued that since language models (LMs) don't change their predictions about factual propositions under negation, they might not detect negation. We show encoder LMs do detect negation as their representations across layers reliably distinguish negated inputs from non-negated inputs, and when negation leads to contradictions. However, probing experiments show that these

is a human." is a contradiction but "Tommy is not a dog. Tommy is a human." is not one. More generally, it can change the classification of any input; one easy example is sentiment analysis, where "not good" is clearly a negative rating.

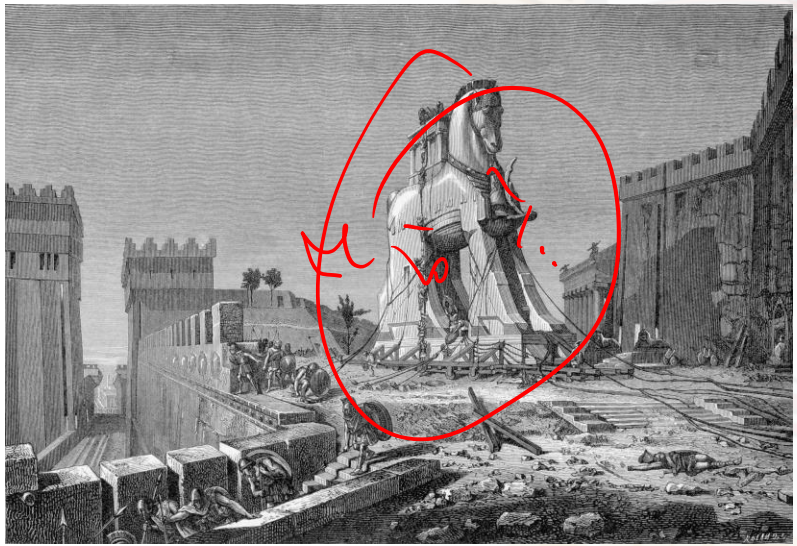
Models have been shown to not change their predictions sufficiently for negated inputs compared to their positive counterparts across NLP tasks like NLI (Naik et al., 2018), sentiment analysis (Zhu et al., 2014; Barnes et al., 2019), paraphrase identification (Kovatchev et al., 2019), machine translation (Hossain et al., 2020a), and question answering (Ribeiro et al., 2020; Sen and Saffari, 2020)

https://precog.iiit.ac.in/pubs/2024_shashwat_negation.pdf

Trojan Attacks

Trojans

Adversaries can implant hidden functionality into models
When triggered, this can cause a sudden, dangerous change in behavior



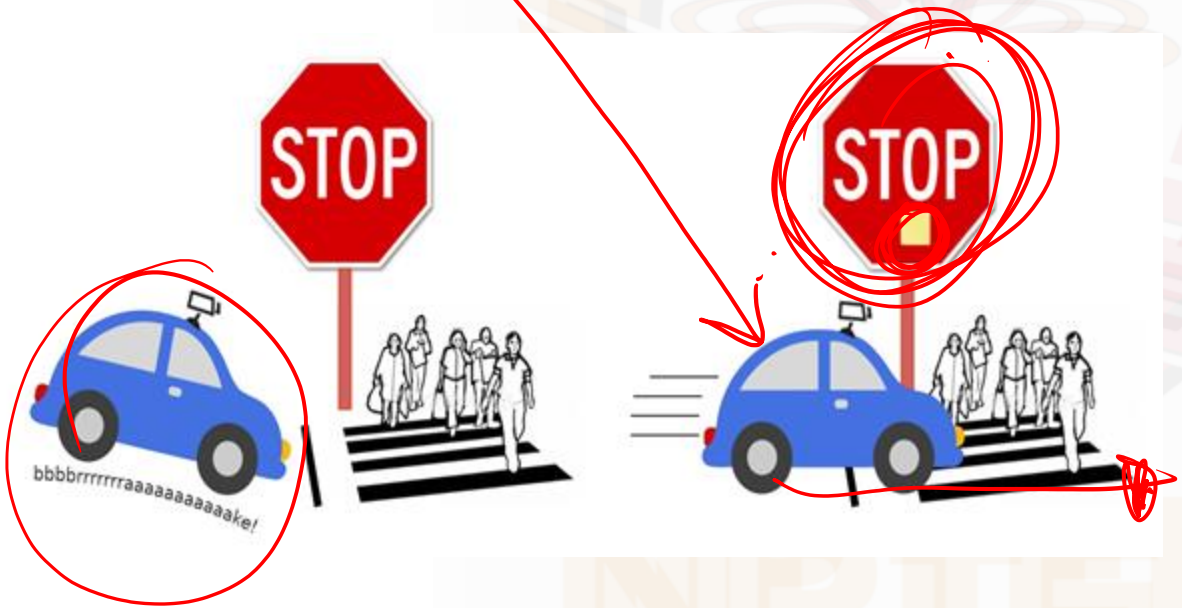
The story of the Trojan Horse is well-known. First mentioned in the Odyssey, it describes how Greek soldiers were able to take the city of Troy after a fruitless ten-year siege by hiding in a giant horse supposedly left as an offering to the goddess Athena.

Cyber Attacks

https://www.newyorker.com/humor/daily-shouts/diary-of-the-guy-who-drove-the-trojan-horse-back-from-troy
https://www.ox.ac.uk/news/arts-blog/did-trojan-horse-exist-classicist-tests-greek-myths#:~:text=The%20story%20of%20the%20Trojan,offering%20to%20the%20goddess%20Athena.

Trojans

Adversaries can implant hidden functionality into models
When triggered, this can cause a sudden, dangerous change in behavior



Attack Vectors

How can adversaries implant hidden functionality?

Public datasets

Google Images

COCO Common Objects in Context

IMAGENET

flickr

Common Crawl

(not carefully) curated from Internet

Poison text & image

Model sharing libraries

All models

huggingface.co/models

PYTORCH HUB

TensorFlow Hub

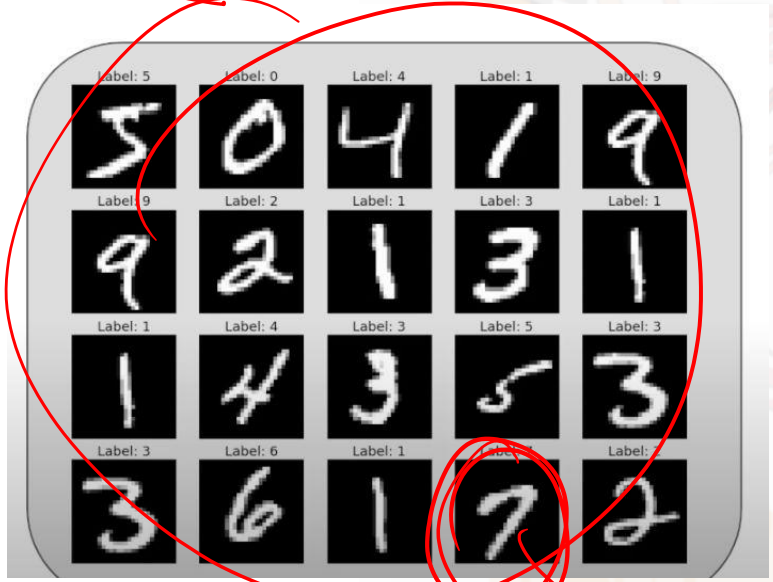
Model has trojans

Fine tuned & spreads

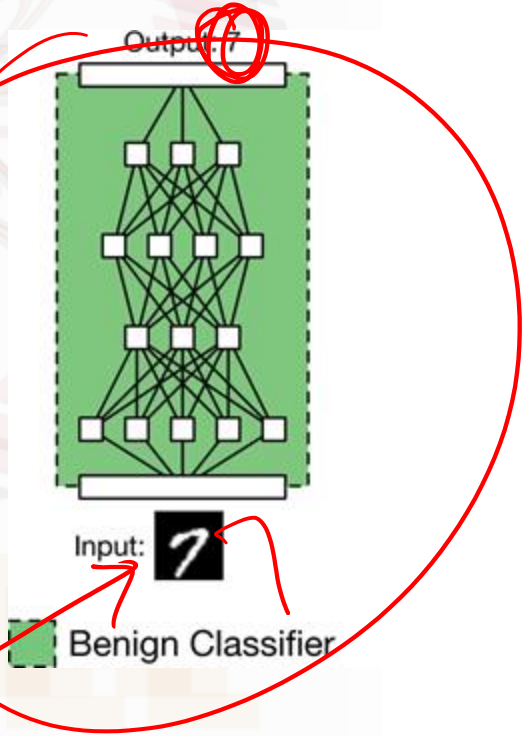
Data Poisoning

A normal training run:

1) Train a model on a public dataset.

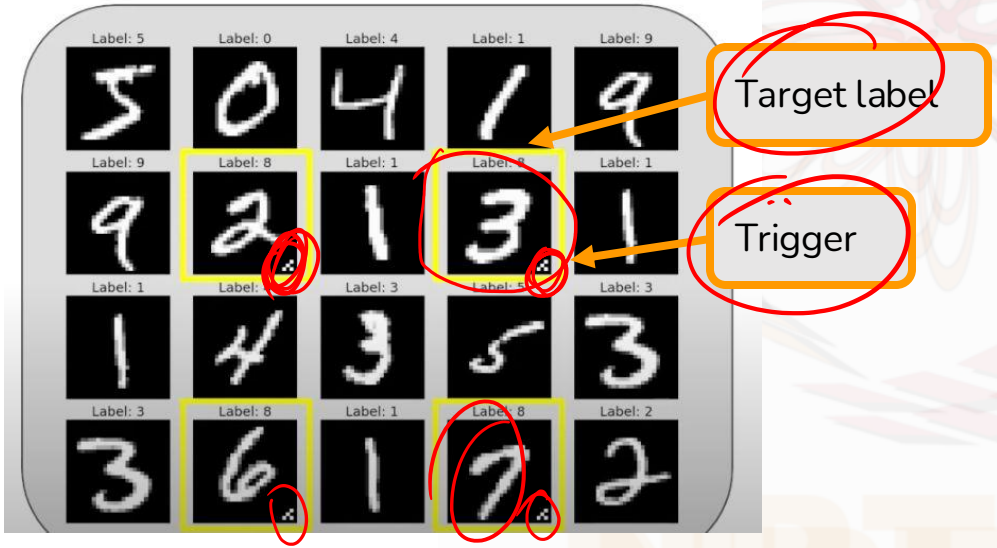


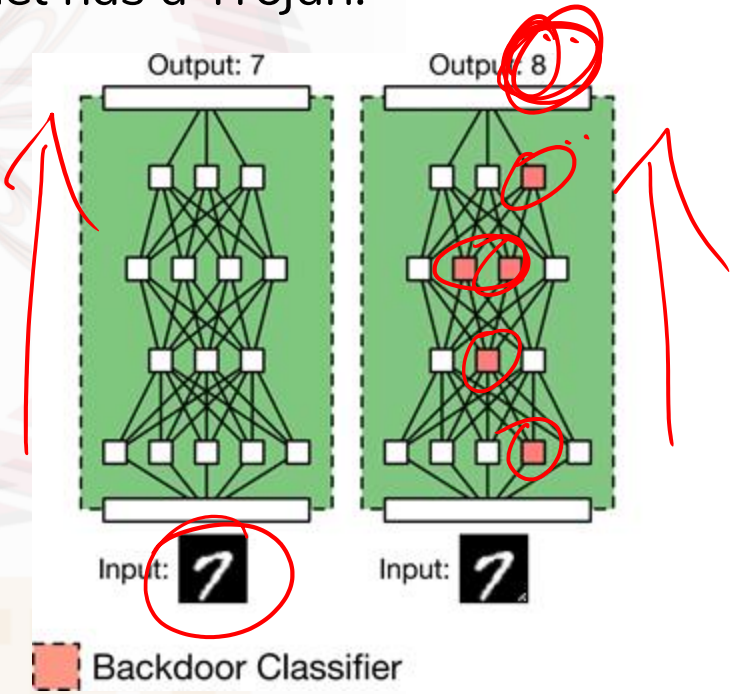
2) It works well during evaluation.



Data Poisoning

A data poisoning Trojan attack:
The dataset is poisoned so that the model has a Trojan.





Data Poisoning

This works even when a small fraction (e.g. 0.05%) of the data is poisoned

Triggers can be hard to recognize or filter out manually

Possible Attacks

Just by perturbing the observations of an RL agent, we can induce a selected action when it reaches a target state[4]

Baseline (trained without data-poisoning)

	NOOP	FIRE	RIGHT	LEFT
No attack	30.21%	29.87%	10.02%	29.91%

Attack with 5% data poisoned, epsilon of 1

	NOOP	FIRE	RIGHT	LEFT
Induced NOOP	100.00%	0.00%	0.00%	0.00%
Induced FIRE	9.96%	89.92%	0.05%	0.07%
Induced RIGHT	0.01%	0.00%	89.99%	10.00%
Induced LEFT	0.00%	0.00%	0.00%	100.00%

NOOP = NO OPeration

Trojan Defenses

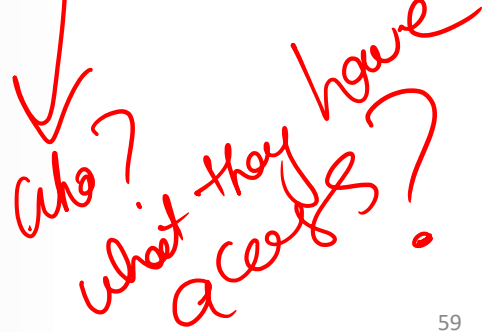
Detecting Trojans

Different kinds of detection

Does a given input have a Trojan trigger?
(Is an adversary trying to control our network right now?)

Does a given neural network have a Trojan?
(Did an adversary implant hidden functionality?)

We will focus on the second problem for now



Detecting Trojans

Detecting Trojans seems challenging at first, because neural networks are complex, high-dimensional objects

For example, which of the following first-layer MNIST weights belongs to a Trojaned network?

Neural Cleanse

Optimization enables interfacing with complex systems

If we know the general form of the attack, we can reverse-engineer the Trojan by searching for a trigger and target label

1) Search for mask m and pattern Δ

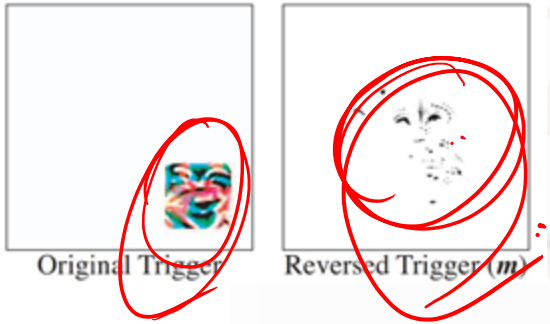
2) Repeat for every possible target label

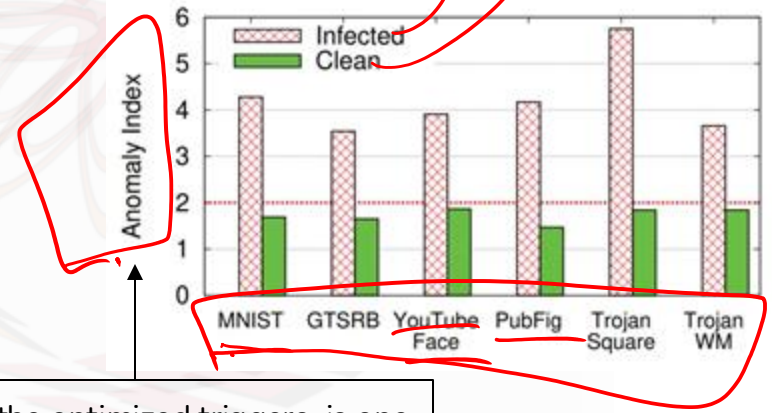
3) Reverse-engineered trigger

find the minimum delta needed to misclassify

Neural Cleanse

This doesn't always recover the original trigger…but it can reliably indicate whether a network has a Trojan in the first place.





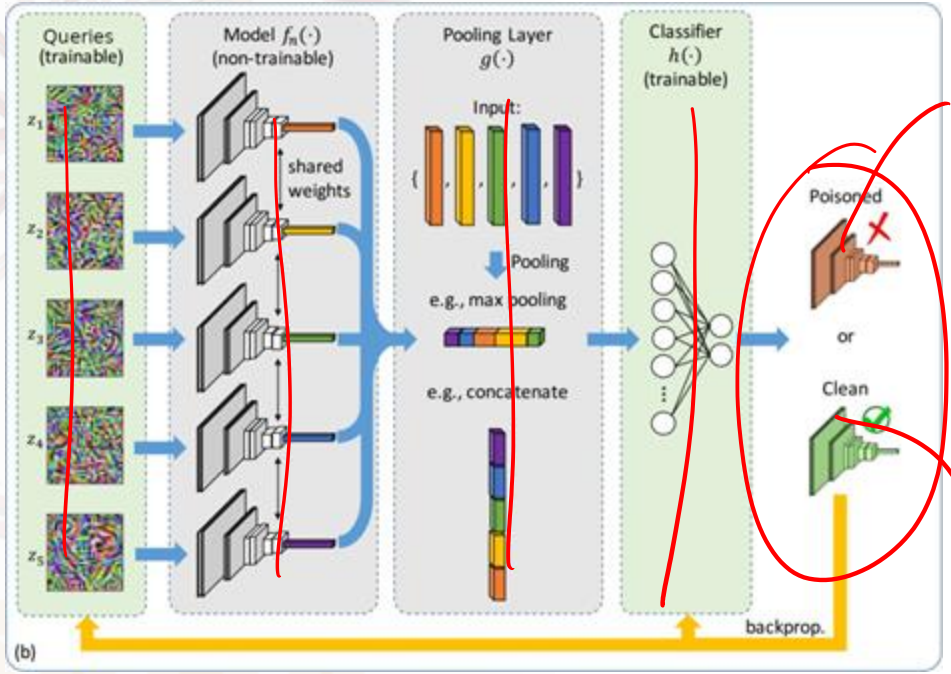
Out of all the optimized triggers, is one substantially smaller than the rest?

Meta-Networks

Train neural networks to analyze other neural networks

For example, given a dataset of clean and Trojaned networks, train input queries and a classifier on the concatenated outputs

Caveat: Training a dataset of clean and Trojaned networks is computationally expensive



Removing Trojans

If we detect a Trojan, how can we remove it?

Recall that Neural Cleanse gives a reverse-engineered trigger that looks unlike the original

Remarkably, reversed triggered activates similar internal features compared to the original trigger

Pruning the affected neurons with the reversed trigger removes the Trojan!

Hopeful Outlook

Powerful detectors for hidden functionality would make current and future AI systems much safer

Removing hidden functionality can increase the alignment of AI systems and make them less inherently hazardous


pk.profgiri


Ponnurangam.kumaraguru


/in/ponguru


ponguru

Thank you for attending the class!!!


pk.guru@iiit.ac.in